1. **Subsetting**

A great feature of numpy arrays is the ability to subset. For instance, if you wanted to know which weights in our weights_kg array are above 90, we could quickly subset it to find out.

```python
import numpy as np
weight_kg = np.array([81.65, 95.25, 92.98, 86.18, 97.52, 88.45])
over_90 = weight_kg > 90
print over_90, weight_kg[over_90]
```

2. **Pyramid of ages: Percentage of men and women by region**

Find the most masculine and the most feminine region using subsetting.

3. **Dot product**

Imagine you are running a bagel shop. You have several recipes of bagels, and for each you store the necessary ingredients in a table (in kg). You also know price of each ingredient: cheese is 15€/kg, ham is 40€/kg, and tomato is 5€/kg. Your task is to compute the price of each bagel.

<table>
<thead>
<tr>
<th></th>
<th>Cheese</th>
<th>Ham</th>
<th>Tomato</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel 1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.05</td>
</tr>
<tr>
<td>Bagel 2</td>
<td>0.05</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Bagel 3</td>
<td>0.2</td>
<td>0</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Price of Bagel 1 is $15 \times 0.1 + 40 \times 0.1 + 5 \times 0.05 = 5.75$. Price of Bagel 2 is $15 \times 0.05 + 40 \times 0.1 + 5 \times 0.1 = 5.25$. Price of Bagel 3 is $15 \times 0.2 + 40 \times 0 + 5 \times 0.1 = 3.5$. We can write the prices as one array $[5.75, 5.25, 3.5]$. We say that this array is a result of a dot product of the array of ingredients and the array of prices.

```python
import numpy as np
ingredients = np.array([[0.1, 0.1, 0.05], [0.05, 0.1, 0.1], [0.2, 0, 0.1]])
prices_ingredients = np.array([15, 40, 5])
prices_bagels = np.dot(ingredients, prices_ingredients)
```

4. **Pyramid of ages: Average age by region**

Assuming that the age of each person in group 0-19 is 10, in group 20-39 is 30, in group 40-59 is 50, in group 60-74 is 67, and in group 70+ is 80, compute the average age in each region. Find regions with highest and smallest average ages.

5. **Pyplot**

**Pyramid of ages: Pie chart of female and male age groups in Alsace**

Here is an example of a pie chart drawn using matplotlib.

```python
import matplotlib.pyplot as plt
import numpy as np

plt.figure(1, figsize=(6,6))
labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
fracs = np.array([15, 30, 45, 10])
plt.pie(fracs,  labels=labels, autopct='%1.1f%%',shadow=True)
plt.title('Raining Hogs and Dogs')
plt.savefig('raining.png')
plt.show()
```

Your task is to draw a pie chart of female and male age groups in Alsace.
Introduction to programming, Lesson 6: numpy

6. Pyramid of ages: Bar plot of male and female populations by ages

Here is an example of a bar chart for programming language usage.

```python
import matplotlib.pyplot as plt
import numpy as np

languages = np.array(['Python', 'C++', 'Java', 'Perl', 'Scala', 'Lisp'])
number_of_users = np.array([10, 8, 6, 4, 2, 1])

plt.title('Programming language usage')
x_pos = np.arange(len(languages))
bc = plt.bar(x_pos-0.4, number_of_users, color='b')
plt.xticks(x_pos, languages)
plt.ylabel('Usage')
plt.savefig('programming_languages.png')
plt.show()
```

Your task is to draw a bar chart of male and female populations by ages. To add a legend to your graph, use `plt.legend((bc_men[0], bc_women[0]), ('Men', 'Women')).`

7. Plots of “scientific” functions

Your task is to draw a function which is equal to \(-x-5\) on \([-10, -5]\), to \(x^2-25\) on \([-5, 4]\), and to \(\sin(x-4)-9\) on \([4, 10]\).

```python
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-5,5,101) # 101 equally spaces ticks from -5 to 5
y = np.sin(x)
plt.plot(x,y)
plt.show()
```

8. (*) 100 Numpy exercises

Go to http://www.labri.fr/perso/nrougier/teaching/numpy.100/. You can solve any problems that you find interesting, see for example 38, 50, 51. Gray boxes contain solutions, so try to find a solution yourself before looking into them.

9. (*) Mandelbrot set

The Mandelbrot set is a fractal which is defined as the set of points \(c\) of the complex plane for which the recurring sequence defined by: \(z_0=0\) and \(z_{n+1}=z_n^2+c\) does not tend towards infinity (in modulus). If we reformulate this without using complex numbers, replacing \(z_n\) by the pair \((x_n, y_n)\) and \(c\) by the pair \((a, b)\) then we get:

\[
x_{n+1}=x_n^2-y_n^2+a \quad \text{and} \quad y_{n+1}=2x_ny_n+b.
\]

It can be shown that as soon as the modulus of \(z_n\) is strictly greater than 2 (\(z_n\) being in algebraic form, when \(x_n^2+y_n^2>2\)), the sequence diverges to infinity, and thus \(c\) is outside the set from Mandelbrot. This allows us to stop computing for points with a modulus strictly greater than 2 and therefore outside the Mandelbrot set. For the points of the Mandelbrot set, i.e., the complex numbers \(c\) for which \(z_n\) does not tend towards infinity, the computation will never reach term, so it must be stopped after a certain number of iterations determined by the program.

Write a script that displays (an approximation of) the entire Mandelbrot.